



C# Avançado

Serialização de Dados


Softblue
cursos online

Tópicos Abordados

- Serialização de objetos
- Grafo de objetos
- Atributos *[Serializable]* e *[NonSerialized]*
- Serialização para o formato binário
 - A classe *BinaryFormatter*
- Serialização para o formato XML
 - A classe *XmlSerializer*
 - Customizando o XML gerado


Serialização de Objetos

- Serializar um objeto é transformá-lo em uma stream de dados de um formato específico



```
graph LR; Objeto((Objeto)) --> Binário[Binário]; Objeto --> XML[XML]; Objeto --> SOAP[SOAP]; Binário --> Destino([Destino<br/>(memória, arquivo,<br/>rede, etc.)]); XML --> Destino; SOAP --> Destino;
```

Atributo *[Serializable]*




- *[Serializable]* pode ser associado a uma classe ou estrutura
 - Indica que instâncias podem ser serializadas

```
[Serializable]
class Aluno
{
    string nome;
    int idade;
}
```

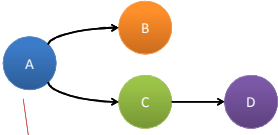
Objetos da classe *Aluno* podem ser serializados

- *[Serializable]* não é herdado pelas subclasses
 - Todas as superclasses precisam definir o atributo
- Fields e auto-implemented properties podem ser serializados

Grafo de Objetos



- Quando um objeto é serializado, todos os objetos dependentes também são




```
graph LR
    A((A)) --> B((B))
    A((A)) --> C((C))
    C((C)) --> D((D))
```

Na serialização de *A*, os objetos *B*, *C* e *D* também são serializados

Todas as classes e estruturas envolvidas devem usar o atributo *[Serializable]*

Caso contrário, uma exceção do tipo *SerializationException* será lançada

Atributo *[NonSerialized]*



- Marca fields ou properties para que sejam excluídos do processo de serialização
 - Quando não existe a necessidade
 - Quando não é possível serializar
 - Redução do espaço de armazenamento

```
[Serializable]
class Aluno
{
    [NonSerialized]
    int tempId;

    string nome;
    int idade;
}
```

tempId não será serializado

Serialização Para o Formato Binário



- Transforma o objeto em um array de bytes
- A classe *BinaryFormatter* é utilizada
 - *System.Runtime.Serialization.Formatters.Binary*

```
BinaryFormatter bf = new BinaryFormatter();
```

- Principais métodos
 - *Serialize()*
 - Transforma o objeto em um array de bytes
 - *Deserialize()*
 - Transforma um array de bytes em um objeto

Método *Serialize()*



```
Aluno a = new Aluno();  
...  
BinaryFormatter bf = new BinaryFormatter();  
using (FileStream s = new FileStream(@"C:\arq.bin", FileMode.Create,  
    FileAccess.Write))  
{  
    bf.Serialize(s, a);  
}
```

Recebe a stream e o objeto
como parâmetros

A serialização ocorre nos fields e auto-implemented
properties, independente da visibilidade

Método *Deserialize()*



```
BinaryFormatter bf = new BinaryFormatter();  
using (FileStream s = new FileStream(@"C:\arq.bin", FileMode.Open,  
    FileAccess.Read))  
{  
    Aluno a = (Aluno)bf.Deserialize(s);  
}
```

Recebe a stream como
parâmetro

Retorna um *object*, então o
casting é necessário

Serialização de Coleções



- É possível também serializar uma coleção de objetos, ao invés de um objeto só
- As principais coleções do C# definem o atributo *[Serializable]*

```
List<Aluno> l = new List<Aluno>();  
...  
BinaryFormatter bf = new BinaryFormatter();  
using (FileStream s = new FileStream(@"C:\arq.bin", FileMode.Create,  
    FileAccess.Write))  
{  
    bf.Serialize(s, l);  
}
```

A lista pode ser usada
como parâmetro

Serialização Para XML



- Transforma o objeto em uma representação no formato XML
- Uso da classe *XmlSerializer*
 - Namespace *System.Xml.Serialization*
- Não é preciso usar o atributo *[Serializable]*
- Apenas fields e auto-implemented properties com visibilidade *public* são serializados
- As classes e estruturas envolvidas no processo devem ter um construtor padrão

A Classe *XmlSerializer*



- Usada no processo de serialização para o formato XML

```
XmlSerializer ser = new XmlSerializer(typeof(Aluno));
```

O tipo de dado a ser
serializado é fornecido
no construtor

- Principais métodos
 - *Serialize()*
 - Transforma o objeto em uma representação de XML
 - *Deserialize()*
 - Transforma uma representação de XML em um objeto

Método *Serialize()*


```

public class Aluno
{
    public string Nome { get; set; }
    public int Idade { get; set; }
    public Turma Turma { get; set; }
}


public class Turma
{
    public int Sala { get; set; }
    public char Letra { get; set; }
}

Aluno a = new Aluno();
...

using (FileStream s = new FileStream(@"C:\arq.xml", FileMode.Create,
    FileAccess.Write))
{
    ser.Serialize(s, a);
}

<?xml version="1.0"?>
<Aluno xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <Nome>Carlos</Nome>
    <Idade>32</Idade>
    <Turma>
        <Sala>13</Sala>
        <Letra>66</Letra>
    </Turma>
</Aluno>

```

Método *Deserialize()*


```


XmlSerializer ser = new XmlSerializer(typeof(Aluno));

using (FileStream s = new FileStream(@"C:\arq.xml", FileMode.Open,
    FileAccess.Read))
{
    Aluno a = (Aluno)ser.Deserialize(s);
}

```

Recebe a stream como parâmetro


Retorna um *object*, então o casting é necessário

Customizando o XML Gerado


- A serialização para XML permite customizar detalhes sobre os dados gerados
- Alguns atributos podem ser usados

Atributo	Descrição
[XmlRoot]	Tag raiz do XML
[XmlElement]	Tag do XML
[XmlAttribute]	Atributo de uma tag do XML
[XmlText]	Texto de uma tag

Exemplo de Customização



```
[XmlRoot(ElementName = "AlunoEscola")]
public class Aluno
{
    [XmlElement(ElementName = "NomeAluno")]
    public string Nome { get; set; }

    [XmlAttribute]
    public int Idade { get; set; }

    [XmlElement]
    public string Descricao { get; set; }
}
```

```
<?xml version="1.0"?>
<AlunoEscola xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" Idade="12">
  <NomeAluno>Pedro</NomeAluno>
  <Descricao>Bom aluno</Descricao>
</AlunoEscola>
```